

FIG. 1

ADDRESS		DATA
0	-----	C0-0
1		C0-1
2		C0-2
3		C0-3
4	-----	C0-4
...		...
n-2	-----	C0-n-2
n-1		C0-n-1
n		C1-0
n+1		C1-1
n+2		C1-2
n+3		C1-3
n+4	-----	C1-4
...		...
n+m-2	-----	C1-m-2
n+m-1		C1-m-1
n+m		C2-0
n+m+1		C2-1
n+m+2		C2-2
n+m+3		C2-3
n+m+4	-----	C2-4
...		...
n+m+l-2	-----	C2-l-2
n+m+l-1		C2-l-1
n+m+l		Blank
n+m+l+1		Blank
n+m+l+2		Blank
n+m+l+3		Blank
n+m+l+4	-----	Blank
...		...
N-2	-----	Blank
N-1	-----	Blank

FIG. 2

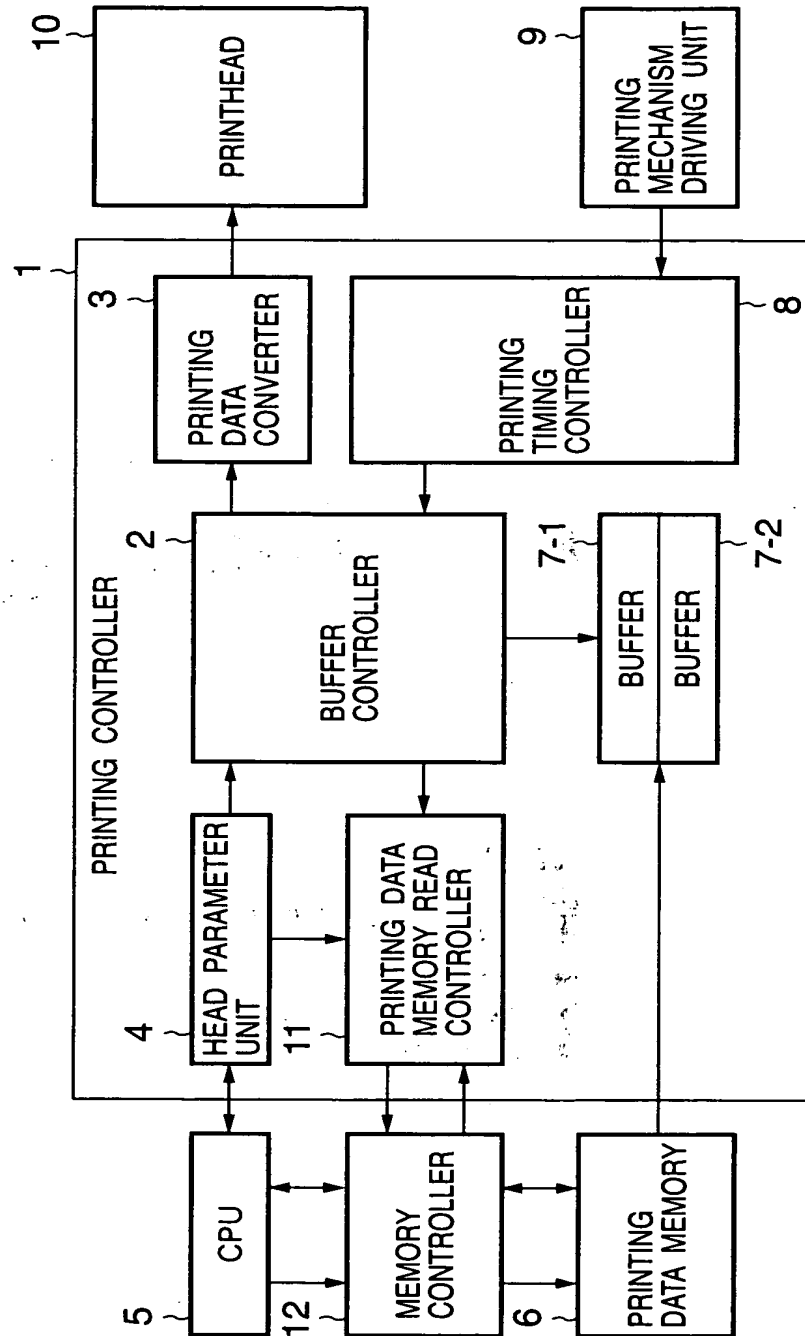


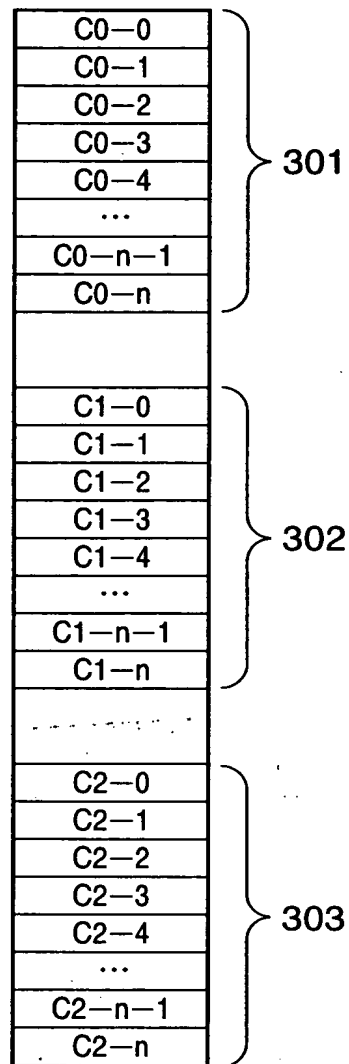
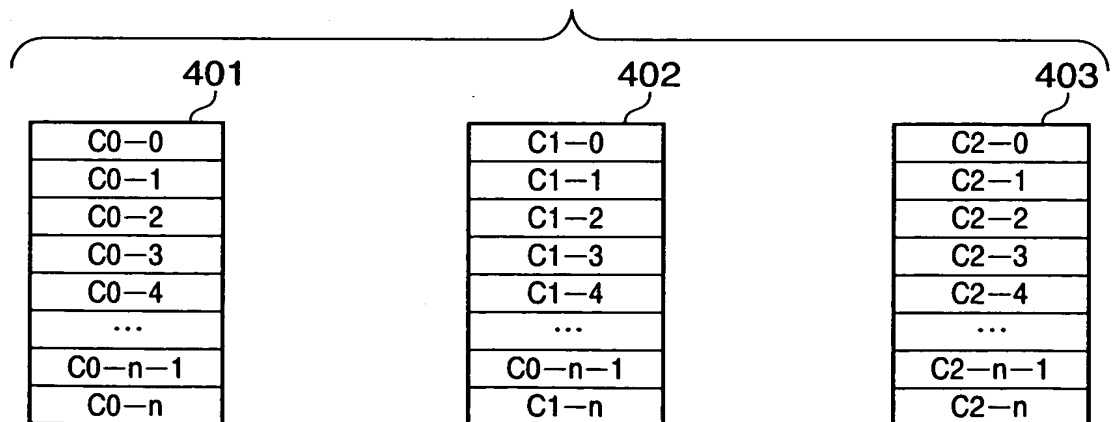
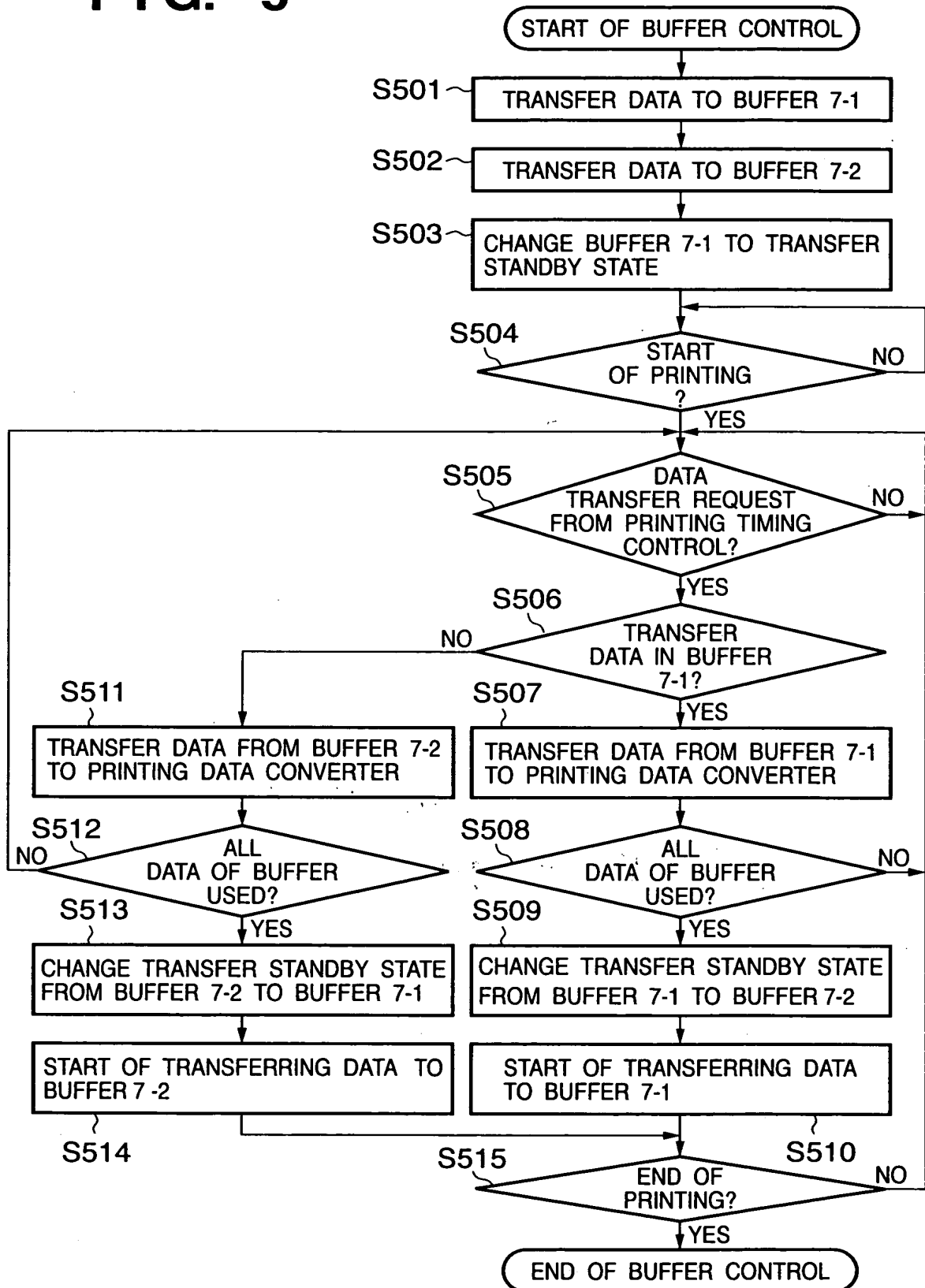
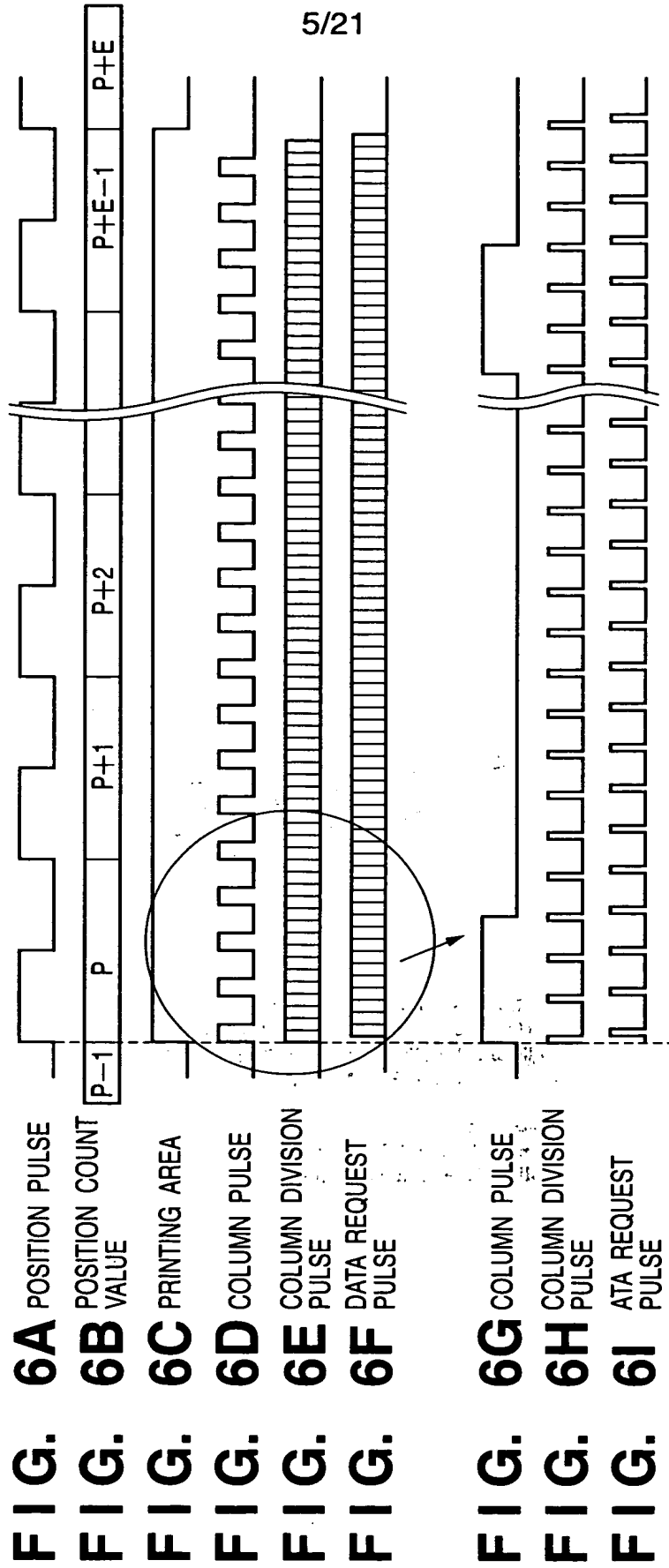
FIG. 3**FIG. 4**

FIG. 5



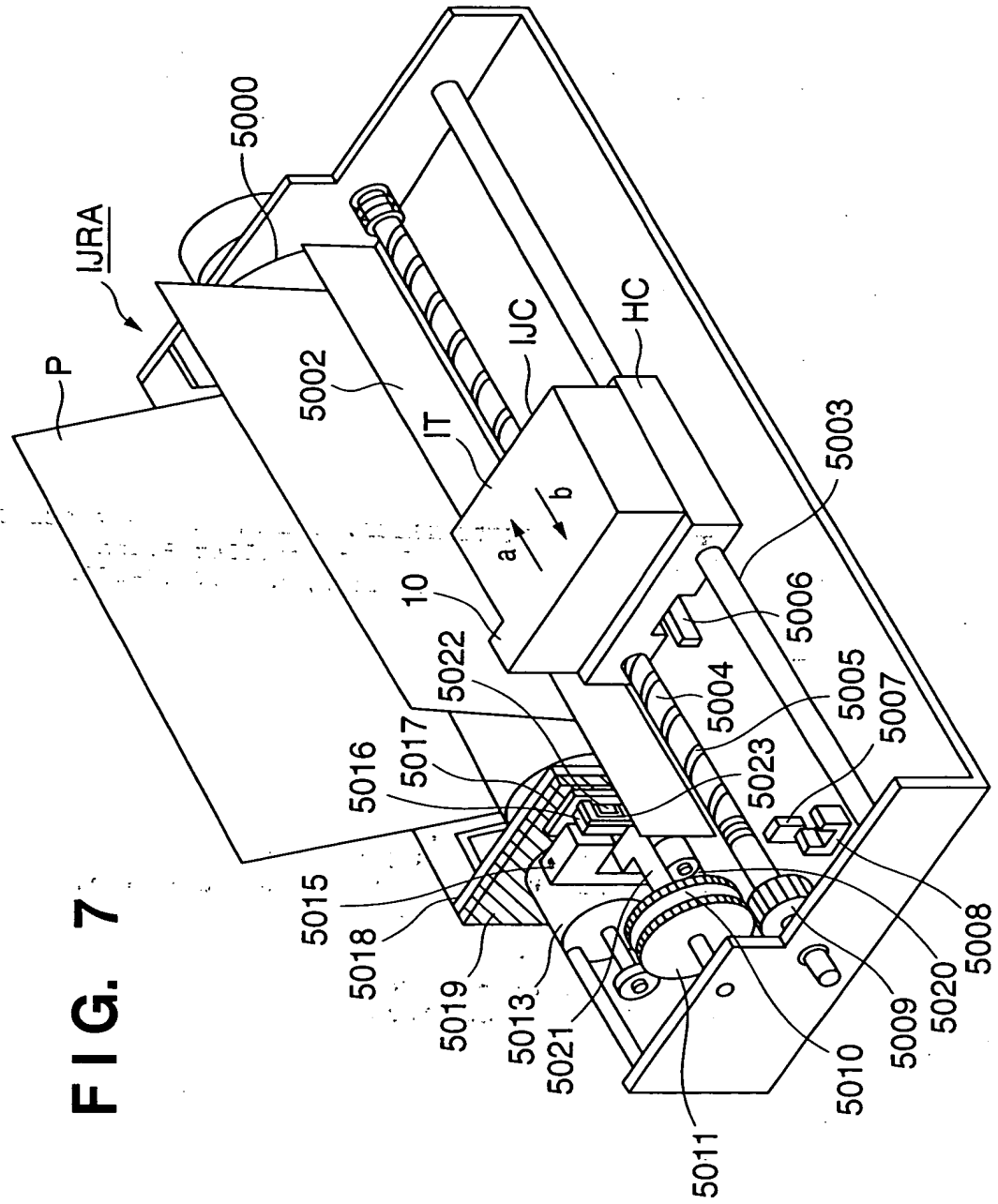


FIG. 7

FIG. 8A

```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity BUFFER CONTROL is
  GENERIC(
    CONSTANT X_MSB :integer:=13; --left range of X num-
    CONSTANT B_NUM :integer:=5;  --data num-1 to Print Cont. per block
    CONSTANT A_MSB :integer:=24;  --left range of address
    CONSTANT A_LSB :integer:=2);  --right range of address

  port(
    rst_n      :IN std_logic;      --asynchronous reset: Low active
    clk        :IN std_logic;      --Pixel clock
    c_max      :IN std_logic_vector(1 DOWNTO 0);      --color num
    c0_num     :IN std_logic_vector(8 DOWNTO 0);      --c0 nozzle num-1
    c1_num     :IN std_logic_vector(8 DOWNTO 0);      --c1 nozzle num-1
    c2_num     :IN std_logic_vector(8 DOWNTO 0);      --c2 nozzle num-1
    c3_num     :IN std_logic_vector(8 DOWNTO 0);      --c3 nozzle num-1
    blk_tbl0_0 :IN std_logic_vector(31 DOWNTO 0);    --block sequence table 0L
    blk_tbl0_1 :IN std_logic_vector(31 DOWNTO 0);    --block sequence table 0H
    n1_diff    :IN std_logic_vector(A_MSB DOWNTO A_LSB); --next line addr diff
    c0_diff    :IN std_logic_vector(A_MSB DOWNTO A_LSB); --c0->c1 address diff
    c1_diff    :IN std_logic_vector(A_MSB DOWNTO A_LSB); --c1->c2 address diff
    c2_diff    :IN std_logic_vector(A_MSB DOWNTO A_LSB); --c2->c3 address diff
    c3_diff    :IN std_logic_vector(A_MSB DOWNTO A_LSB); --c3->c0 address diff
    bm_strt_adr :IN std_logic_vector(A_MSB DOWNTO A_LSB); --bitmap start addr
    Bsync      :IN std_logic;      --block sync pulse
    print_e    :IN std_logic;      --print enable
    BE         :OUT std_logic_vector(3 DOWNTO 0);    --heat block
    data       :OUT std_logic_vector(B_NUM DOWNTO 0); --heat data
    init       :IN std_logic;      --initialize from arbter
    r_na       :IN std_logic;      --Next read cycle Address request of bitmap
    r_valid    :IN std_logic;      --bitmap data valid
    r_req      :OUT std_logic;     --bitmap read request
    r_adr      :OUT std_logic_vector(A_MSB DOWNTO A_LSB); --bitmap address
    dat_0      :IN std_logic_vector(31 DOWNTO 0);    --SRAM0 data
    dat_1      :IN std_logic_vector(31 DOWNTO 0);    --SRAM1 data
    adr_0      :OUT std_logic_vector(8 DOWNTO 0);    --SRAM0 address
    adr_1      :OUT std_logic_vector(8 DOWNTO 0);    --SRAM1 address
    we_0       :OUT std_logic;     --SRAM0 write enable
    we_1       :OUT std_logic;     --SRAM1 write enable
  );
end BUFFER CONTROL;

```

FIG. 8B

ARCHITECTURE RTL OF BUFFER CONTROL IS

```

constant B4_0      :std_logic_vector(3 downto 0)      :=(others=>'0');
constant B4_1      :std_logic_vector(3 downto 0)      :=(others=>'1');
constant B5_0      :std_logic_vector(4 downto 0)      :=(others=>'0');
constant B7_0      :std_logic_vector(6 downto 0)      :=(others=>'0');
constant B9_0      :std_logic_vector(8 downto 0)      :=(others=>'0');
constant X_11_0    :std_logic_vector(X_MSB downto 11) :=(others=>'0');
constant X_12_0    :std_logic_vector(X_MSB downto 12) :=(others=>'0');
signal ini         :std_logic;                        --counter initialize
signal x_count     :std_logic_vector(X_MSB downto 0);
signal Bsync_1     :std_logic;                        --block sync. delay 1
signal Bsync_2     :std_logic;                        --block sync. delay 2
signal Bsync_3     :std_logic;                        --block sync. delay 3
signal Bsync_p     :std_logic;                        --block sync. rising edge
signal b_count     :std_logic_vector(3 downto 0);      --block sync. count
signal column_end  :std_logic;
signal prt_ena_1   :std_logic;                        --print enable delay 1
signal prt_ena_2   :std_logic;                        --print enable delay 2
signal prt_ena_3   :std_logic;                        --print enable delay 3
signal Hsync_p     :std_logic;                        --print enable rising edge
signal color_end   :std_logic;                        --nozzle boundary of color
signal color_base  :std_logic_vector(8 downto 0);     --color boundary
signal buf_valid   :std_logic;                        --buffer is valid
signal buf_valid_1 :std_logic;                        --buffer address is valid
signal buf_valid_2 :std_logic;                        --buffer output is valid
signal n_count     :std_logic_vector(4 downto 0);     --nozzle num in block
signal color       :std_logic_vector(1 downto 0);     --color num -1
signal color_1     :std_logic_vector(1 downto 0);     --color for SRAM out :delay 1
signal n_num_div16 :std_logic_vector(8 downto 4);     --c_num/16
signal block_end   :std_logic;
signal c_num       :std_logic_vector(8 downto 0);     --color nozzle num-1
signal b_r_adr     :std_logic_vector(8 downto 0);     --buffer read address
signal r_adr_tmp   :std_logic_vector(A_MSB downto A_LSB); --EXT RAM address
signal r_color     :std_logic_vector(1 downto 0);     --color for EXT RAM read
signal bf_chg      :std_logic;                        --read buffer change
signal r_count     :std_logic_vector(8 downto 0);     --EXT RAM read counter
signal r_c_count   :std_logic_vector(8 downto 0);     --EXT RAM color num counter
signal r_c_end     :std_logic;                        --EXT RAM color end
signal r_c_num     :std_logic_vector(8 downto 0);     --color num-1 for EXT RAM
signal r_v_count   :std_logic_vector(8 downto 0);     --EXT RAM read valid counter
signal r_req_tmp   :std_logic;                        --bitmap read request
signal r_req_stack :std_logic_vector(2 downto 0);     --EXT RAM read req stack
signal r_ini_end   :std_logic;                        --1st buffer fill req end
signal w_ini_end   :std_logic;                        --1st buffer fill write end
signal r_end       :std_logic;                        --EXT RAM read end
signal w_end       :std_logic;                        --buffer write end
signal data        :std_logic_vector(B_NUM downto 0); --heat data
signal bn          :std_logic_vector(3 downto 0);     --nozzle block number
signal x_pst       :std_logic_vector(X_MSB downto 0); --x position for heat data
signal bf0_rd      :std_logic_vector(3 downto 0);     --buffer_0 read for odd nozzle
signal bf0_rd_all  :std_logic;                        --all nozzle read buffer_0
signal bf1_rd_all  :std_logic;                        --all nozzle read buffer_1
signal bf0_read    :std_logic;                        --buffer_0 read
signal bf0_read_1  :std_logic;                        --buffer_0 read delay 1
signal h_dat       :std_logic;                        --heat data
signal tb_sel      :std_logic_vector(1 downto 0);     --block seq. table select
signal bn_seq      :std_logic_vector(63 downto 0);    --block sequence
signal x_diff      :std_logic_vector(A_MSB downto A_LSB);
signal h_dat_s     :std_logic;                        --heat data from SRAM
signal bf0_wite    :std_logic;                        --buffer_0 write
signal c_max_tmp   :std_logic_vector(1 downto 0);     --color num for yobito
signal buf_v_ini   :std_logic;                        --for buffer initial read
signal buf_vld_1   :std_logic;                        --buffer address is valid
signal pst_1       :std_logic_vector(4 downto 0);     --x pos for read address

```


9/21

FIG. 8C

BEGIN

ini <= NOT(print_e) OR init ;
buf_vld_1 <= buf_valid AND buf_valid_1 ;

Bsync_dly : PROCESS (clk, rst_n)
BEGIN

IF (rst_n='0') THEN

Bsync_1 <='0' ;
Bsync_2 <='0' ;
Bsync_3 <='0' ;
prt_ena_1 <='0' ;
prt_ena_2 <='0' ;
prt_ena_3 <='0' ;

ELSIF clk='1' AND clk'event THEN

IF ini='1' OR ena='0' THEN

Bsync_1 <='0' ;
Bsync_2 <='0' ;
Bsync_3 <='0' ;
prt_ena_1 <='0' ;
prt_ena_2 <='0' ;
prt_ena_3 <='0' ;

ELSE

Bsync_1 <=Bsync ;
Bsync_2 <=Bsync_1 ;
Bsync_3 <=Bsync_2 ;
prt_ena_1 <=print_e ;
prt_ena_2 <=prt_ena_1 ;
prt_ena_3 <=prt_ena_2 ;

END IF ;

END IF ;

END PROCESS Bsync_dly ;

Bsync_pulse : PROCESS (Bsync_2, Bsync_3)

BEGIN

IF Bsync_2='1' AND Bsync_3='0' THEN
Bsync_p <='1' ;

ELSE

Bsync_p <='0' ;

END IF ;

END PROCESS Bsync_pulse ;

Hsync_pulse : PROCESS (prt_ena_2, prt_ena_3)

BEGIN

IF prt_ena_2='1' AND prt_ena_3='0' THEN
Hsync_p <='1' ;

ELSE

Hsync_p <='0' ;

END IF ;

END PROCESS Hsync_pulse ;

--for buffer read

clr_bas_gen : PROCESS (clk, rst_n)

BEGIN

IF (rst_n='0') THEN

color_base <=(others =>'0') ;

ELSIF clk='1' AND clk'event THEN

IF init='1' OR block_end='1' THEN

color_base <=(others =>'0') ;

ELSIF color_end='1' THEN

color_base <=color_base+c_num+'1' ;

END IF ;

END IF ;

END PROCESS clr_bas_gen ;

801

10/21

FIG. 8D

```

num_cnt : PROCESS(clk, rst_n)--color access counter of each block
BEGIN
    IF (rst_n='0') THEN
        n_count<=(others=>'0');
    ELSIF clk='1' AND clk'event THEN
        IF buf_vld_1='0' OR color_end='1' THEN
            n_count<=(others=>'0');
        ELSE
            n_count<=n_count+'1';
        END IF;
    END IF;
END PROCESS num_cnt;

end_ncnt_det : PROCESS (n_count, n_num_div16, buf_vld_1)
BEGIN
    IF n_count=n_num_div16 THEN
        color_end<=buf_vld_1;
    ELSE
        color_end<='0';
    END IF;
END PROCESS end_ncnt_det;

color_cnt : PROCESS (clk, rst_n)
BEGIN
    IF (rst_n='0') THEN
        color<=(others=>'0');
    ELSIF clk='1' AND clk'event THEN
        IF init='1' OR block_end='1' THEN
            color<=(others=>'0');
        ELSIF color_end='1' THEN
            color<=color+'1';
        END IF;
    END IF;
END PROCESS color_cnt;

end_blk_det : PROCESS (color, c_max_tmp, color_end)
BEGIN
    IF color=c_max_tmp THEN
        block_end<=color_end;
    ELSE
        block_end<='0';
    END IF;
END PROCESS end_blk_det;

div16_mux : PROCESS (color, c0_num, c1_num, c2_num, c3_num, c_num)
BEGIN
    CASE color IS
        WHEN "00" => c_num<=c0_num;
        WHEN "01" => c_num<=c1_num;
        WHEN "10" => c_num<=c2_num;
        WHEN others => c_num<=c3_num;
    END CASE;
    n_num_div16<=c_num(8 DOWNTO 4);
END PROCESS div16_mux;

buf_vld_gen : PROCESS (clk, rst_n)
BEGIN
    IF (rst_n='0') THEN
        buf_valid<='0';
    ELSIF clk='1' AND clk'event THEN
        IF init='1' OR block_end='1' THEN
            buf_valid<='0';
        ELSIF Bsync_p='1' THEN
            buf_valid<='1';
        ELSE
            IF w_ini_end='0' AND w_end='1' THEN
                buf_valid<='1';
            END IF;
        END IF;
    END IF;
END PROCESS buf_vld_gen;

```

11/21

FIG. 8E

```

b_cnt : PROCESS (rst_n, clk)
BEGIN
    IF rst_n='0' THEN
        b_count<=(others =>'0');
    ELSIF clk='1' AND clk'event THEN
        IF ini='1' OR Hsync_p='1' OR column_end='1' THEN
            b_count<=(others =>'0');
        ELSIF Bsync_p='1' THEN
            b_count<=b_count+'1';          --count up
        END IF;
    END IF;
END PROCESS b_cnt;

end_Block_det : PROCESS (dir, bs_auto, b_count, Bsync_p)
BEGIN
    IF dir='0' OR bs_auto='0' THEN
        IF b_count=B4_1 THEN
            column_end<=Bsync_p;
        ELSE
            column_end<='0';
        END IF;
    ELSE
        IF b_count=B4_0 THEN
            column_end<=Bsync_p;
        ELSE
            column_end<='0';
        END IF;
    END IF;
END PROCESS end_Block_det;

x_cnt : PROCESS (rst_n, clk)
BEGIN
    IF rst_n='0' THEN
        x_count<=(others =>'0');
    ELSIF clk='1' AND clk'event THEN
        IF ini='1' OR Hsync_p='1' THEN
            x_count<=(others =>'0');
        ELSIF column_end='1' THEN
            x_count<=x_count+'1';
        END IF;
    END IF;
END PROCESS x_cnt;

x_pst<=x_count+x_off;

b_r_adr_gen : PROCESS (n_count, color_base, bn, x_pst)
variable b_r_adr_tmp : std_logic_vector(8 downto 0);
BEGIN
    b_r_adr_tmp :=color_base+(n_count(4 downto 1)& bn & n_count(0));
    b_r_adr <=b_r_adr_tmp(7 downto 0)& x_pst(5);
END PROCESS b_r_adr_gen;

```

12/21

FIG. 8F

```
bf_vld_dly : PROCESS (clk, rst_n)
BEGIN
    IF (rst_n='0') THEN
        buf_valid_1<='0';
        buf_valid_2<='0';
    ELSIF clk='1' AND clk'event THEN
        IF init='1' THEN
            buf_valid_1<='0';
            buf_valid_2<='0';
        ELSE
            buf_valid_1<=buf_valid;
            buf_valid_2<=buf_vld_1;
        END IF;
    END IF;
END PROCESS bf_vld_dly;

color_dly : PROCESS (clk, rst_n)
BEGIN
    IF (rst_n='0') THEN
        color_1<=(others=>'0');
    ELSIF clk='1' AND clk'event THEN
        color_1<=color;
    END IF;
END PROCESS color_dly;

bf0_read<=NOT x_pst(5);

bf0_read_dly : PROCESS (rst_n, clk)
BEGIN
    IF rst_n='0' THEN
        bf0_read_1<='0';
    ELSIF clk='1' AND clk'event THEN
        bf0_read_1<=bf0_read;
    END IF;
END PROCESS bf0_read_dly;

bn_seq<=blk_tbl0_0 & blk_tbl0_1;
bn_dly : PROCESS (rst_n, clk)
variable bn_sel : std_logic_vector(3 downto 0);
variable lsb : integer range 0 to 63;
BEGIN
    IF rst_n='0' THEN
        bn<=(others=>'0');
    ELSIF clk='1' AND clk'event THEN
        IF buf_valid='1' AND buf_valid_1='0' THEN
            bn_sel := NOT (b_count);
            lsb := CONV_INTEGER ('0' & bn_sel & "00");
            FOR i IN 0 TO 3 LOOP
                bn(i)<=bn_seq(i+lsb);
            END LOOP;
        END IF;
    END IF;
END PROCESS bn_dly;
```

FIG. 8G

```

bf0_rd_d : PROCESS (rst_n, clk)
BEGIN
    IF rst_n='0' THEN
        bf0_rd <= (others => '0');
    ELSIF clk='1' AND clk'event THEN
        IF init='1' THEN
            bf0_rd <= (others => '0');
        ELSIF buf_vld_1='1' THEN
            bf0_rd (CONV_INTEGER(color)) <= bf0_read;
        END IF;
    END IF;
END PROCESS bf0_rd_d;

bf_rd_all_det : PROCESS (c_max_tmp, e_bf0_rd, o_bf0_rd)
BEGIN
    bf0_rd_all <= '0';
    bf1_rd_all <= '0';
    CASE c_max_tmp IS
        WHEN "00" =>
            IF bf0_rd(0)='1' THEN
                bf0_rd_all <= '1';
            END IF;
            IF bf0_rd(0)='0' THEN
                bf1_rd_all <= '1';
            END IF;
        WHEN "01" =>
            IF bf0_rd(1 DOWNTO 0)="11" THEN
                bf0_rd_all <= '1';
            END IF;
            IF bf0_rd(1 DOWNTO 0)="00" THEN
                bf1_rd_all <= '1';
            END IF;
        WHEN "10" =>
            IF bf0_rd(2 DOWNTO 0)="111" THEN
                bf0_rd_all <= '1';
            END IF;
            IF bf0_rd(2 DOWNTO 0)="000" THEN
                bf1_rd_all <= '1';
            END IF;
        WHEN others =>
            IF bf0_rd(3 DOWNTO 0)="1111" THEN
                bf0_rd_all <= '1';
            END IF;
            IF bf0_rd(3 DOWNTO 0)="0000" THEN
                bf1_rd_all <= '1';
            END IF;
    END CASE;
END PROCESS bf_rd_all_det;

bf_write_det : PROCESS (rst_n, clk)
BEGIN
    IF rst_n='0' THEN
        bf0_wite <= '1';
    ELSIF clk='1' AND clk'event THEN
        IF init='1' THEN
            bf0_wite <= '1';
        ELSIF bf_chg_1='1' THEN
            bf0_wite <= NOT bf0_wite;
        END IF;
    END IF;
END PROCESS bf_write_det;

bf_chg_gen : PROCESS (Hsync_p, w_ini_end, w_end, bf0_wite, bf0_rd_all, bf1_rd_all)
variable ini_chg : std_logic;
BEGIN
    ini_chg := (NOT w_ini_end AND w_end);
    IF bf0_wite='1' THEN
        bf_chg <= ini_chg OR (w_end AND bf0_rd_all);
    ELSE
        bf_chg <= ini_chg OR (w_end AND bf1_rd_all);
    END IF;
END PROCESS bf_chg_gen;

```

FIG. 8H

```

pst_dly : PROCESS (clk, rst_n)
BEGIN
  IF (rst_n='0') THEN
    pst_1<=(others =>'1');
  ELSIF clk='1' AND clk'event THEN
    pst_1<=NOT x_pst(4 downto 0);
  END IF;
END PROCESS pst_dly;

h_dat_s_mux : PROCESS (pst_1, bf0_read_1, dat_0, dat_1)
variable   pst           : integer range 0 to 31;
BEGIN
  pst :=CONV_INTEGER ('0' & pst_1);

  IF bf0_read_1='1' THEN
    h_dat_s<=dat_0(pst);
  ELSE
    h_dat_s<=dat_1(pst);
  END IF;
END PROCESS h_dat_s_mux;

h_dat<=h_dat_s;

out_pack : PROCESS (clk, rst_n)
BEGIN
  IF (rst_n='0') THEN
    data<=(others =>'0');
  ELSIF clk='1' AND clk'event THEN
    IF buf_valid_2='1' AND(print_e='1' OR buf_v_ini='1') THEN
      data(0)<=h_dat;
      FOR i IN TO B_NUM LOOP
        data(i)<=data(i-1);
      END LOOP;
    END IF;
  END IF;
END PROCESS out_pack;

```

FIG. 8I

```

r_v_cnt : PROCESS (clk, rst_n)
BEGIN
  IF (rst_n='0') THEN
    r_v_count<=(others =>'0');
  ELSIF clk='1' AND clk'event THEN
    IF init='1' OR bf_chg='1' THEN
      r_v_count<=(others =>'0');
    ELSIF r_valid='1' THEN
      r_v_count<=r_v_count+'1';
    END IF;
  END IF;
END PROCESS r_v_cnt;

end_w_ini : PROCESS (clk, rst_n)
BEGIN
  IF (rst_n='0') THEN
    w_ini_end<='0';
  ELSIF clk='1' AND clk'event THEN
    IF init='1' THEN
      w_ini_end<='0';
    ELSIF w_end='1' THEN
      w_ini_end<='1';
    END IF;
  END IF;
END PROCESS end_w_ini;

w_end_det : PROCESS (r_req_tmp, r_req_stack)
BEGIN
  IF r_req_tmp='0' AND r_req_stack="000" THEN
    w_end<='1';
  ELSE
    w_end<='0';
  END IF;
END PROCESS w_end_det;

bf_adr_mux : PROCESS (r_valid, bf0_wite, print_e, yp_yobi, b_r_adr, r_v_count)
BEGIN
  IF r_valid='1' THEN
    IF bf0_wite='0' THEN
      adr_0<=b_r_adr;
      adr_1<=r_v_count;
      we_0<='0';
      we_1<='1';
    ELSE
      adr_0<=r_v_count;
      adr_1<=b_r_adr;
      we_0<='1';
      we_1<='0';
    END IF;
  ELSE
    we_0<='0';
    we_1<='0';
    adr_0<=b_r_adr;
    adr_1<=b_r_adr;
  END IF;
END PROCESS bf_adr_mux;

bf_v_ini_det : PROCESS (clk, rst_n)
BEGIN
  IF (rst_n='0') THEN
    buf_v_ini<='0';
  ELSIF clk='1' AND clk'event THEN
    IF init='1' OR block_end='1' THEN
      buf_v_ini<='0';
    ELSE
      IF w_ini_end='0' AND w_end='1' THEN
        buf_v_ini<='1';
      END IF;
    END IF;
  END IF;
END PROCESS bf_v_ini_det;

```

FIG. 8J

```

cmax_det : PROCESS (c_max)
BEGIN
    IF c_max="11" THEN
        c_max_tmp<="11";
    ELSE
        c_max_tmp<=c_max;
    END IF;
END PROCESS cmax_det;

r_req_gen : PROCESS (clk, rst_n)
BEGIN
    IF (rst_n='0') THEN
        r_req_tmp<='0';
    ELSIF clk='1' AND clk'event THEN
        IF init='1' OR bf_chg='1' THEN
            r_req_tmp<='1';
        ELSIF r_end='1' THEN
            r_req_tmp<='0';
        END IF;
    END IF;
END PROCESS r_req_gen;

r_req_stck : PROCESS (clk, rst_n)
BEGIN
    IF (rst_n='0') THEN
        r_req_stck<=(others =>'1');
    ELSIF clk='1' AND clk'event THEN
        IF init='1' THEN
            r_req_stck<=(others =>'0');
        ELSIF r_na='1' THEN
            IF r_valid='0' THEN
                r_req_stck(2 downto 1)<=r_req_stck(2 downto 1)+'1';
            ELSE
                r_req_stck<=r_req_stck+'1';
            END IF;
        ELSIF r_valid='1' THEN
            r_req_stck<=r_req_stck-'1';
        END IF;
    END IF;
END PROCESS r_req_stck;

end_r_ini : PROCESS (clk, rst_n)
BEGIN
    IF (rst_n='0') THEN
        r_ini_end<='0';
    ELSIF clk='1' AND clk'event THEN
        IF init='1' THEN
            r_ini_end<='0';
        ELSIF r_end='1' THEN
            r_ini_end<='1';
        END IF;
    END IF;
END PROCESS end_r_ini;

r_end_det : PROCESS (r_color, c_max_tmp, r_c_end)
BEGIN
    IF r_color=c_max_tmp AND r_c_end='1' THEN
        r_end<='1';
    ELSE
        r_end<='0';
    END IF;
END PROCESS r_end_det;

r_adr<=r_adr_tmp;
r_req<=r_req_tmp;
BE<=o_bn;

```


FIG. 8K

```

--for EXT RAM read
r_ad_gen : PROCESS (clk, rst_n)
BEGIN
  IF (rst_n='0') THEN
    r_adr_tmp<=(others =>'0');
  ELSIF clk='1' AND clk'event THEN
    IF init='1' THEN
      r_adr_tmp<=bm_strt_adr;
    ELSIF r_na='1' THEN
      IF r_c_end='1' THEN
        r_adr_tmp<=r_adr_tmp+x_diff;
      ELSE
        r_adr_tmp<=r_adr_tmp+n1_diff;
      END IF;
    END IF;
  END IF;
END PROCESS r_ad_gen;

dif_sel : PROCESS (r_color, c0_diff, c1_diff, c2_diff, c3_diff, c0_num, c1_num, c2_num, c3_num)
BEGIN
  CASE r_color IS
    WHEN "00"      =>    x_diff<=c0_diff;    r_c_num<=c0_num;
    WHEN "01"      =>    x_diff<=c1_diff;    r_c_num<=c1_num;
    WHEN "10"      =>    x_diff<=c2_diff;    r_c_num<=c2_num;
    WHEN others     =>    x_diff<=c3_diff;    r_c_num<=c3_num;
  END CASE
END PROCESS dif_sel;

c_cnt : PROCESS (clk, rst_n) --EXT RAM read counter for color
BEGIN
  IF (rst_n='0') THEN
    r_c_count<=(others =>'0');
  ELSIF clk='1' AND clk'event THEN
    IF init='1' THEN
      r_c_count<=(others =>'0');
    ELSIF r_c_end='1' THEN
      r_c_count<=(others =>'0');
    ELSIF r_na='1' THEN
      r_c_count<=r_c_count+'1';
    END IF;
  END IF;
END PROCESS c_cnt;

r_col_cnt : PROCESS (clk, rst_n)
BEGIN
  IF (rst_n='0') THEN
    r_color<=(others =>'0');
  ELSIF clk='1' AND clk'event THEN
    IF init='1' OR bf_chg='1' THEN
      r_color<=(others =>'0');
    ELSIF r_c_end='1' THEN
      r_color<=r_color+'1';
    END IF;
  END IF;
END PROCESS r_col_cnt;

end_col_det : PROCESS (r_c_count, r_c_num, r_na)
BEGIN
  IF r_count=r_c_num THEN
    r_c_end<=r_na;
  ELSE
    r_c_end<='0';
  END IF;
END PROCESS end_col_det;

END RTL;

```

FIG. 9

Parameter Name	Parameter Address	Parameter Value
c_num	0(Hex)	2
c0_n_num	4(Hex)	n
c1_n_num	8(Hex)	m
c2_n_num	C(Hex)	l
c0_diff	10(Hex)	1
c1_diff	14(Hex)	1
c2_diff	18(Hex)	FFFFFFC1
bm_strt_adr	1C(Hex)	100
nl_diff	20(Hex)	40

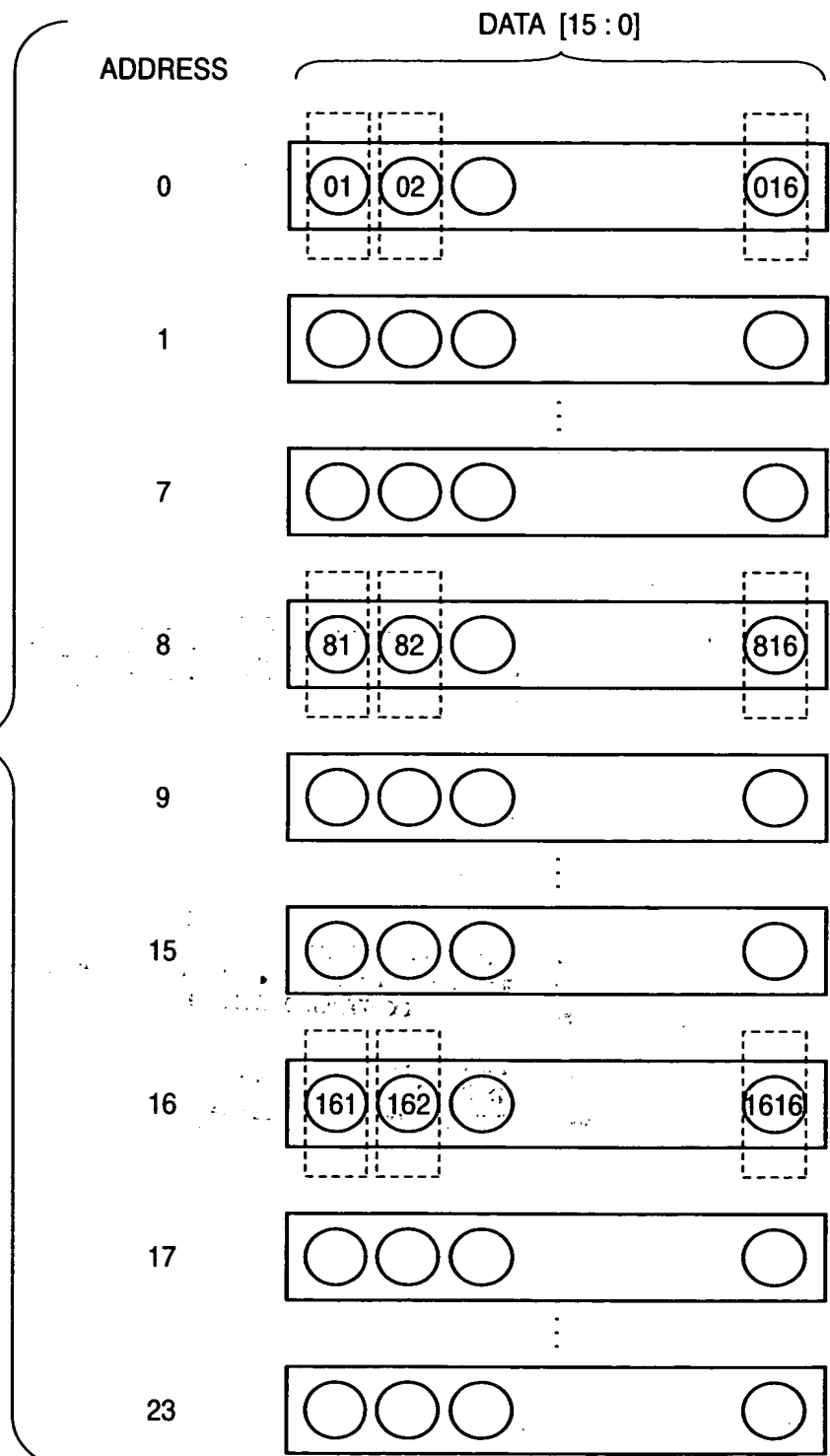
FIG. 10

FIG. 11

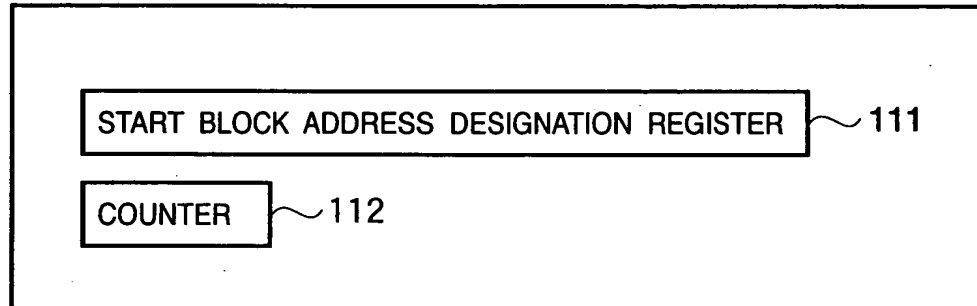


FIG. 12

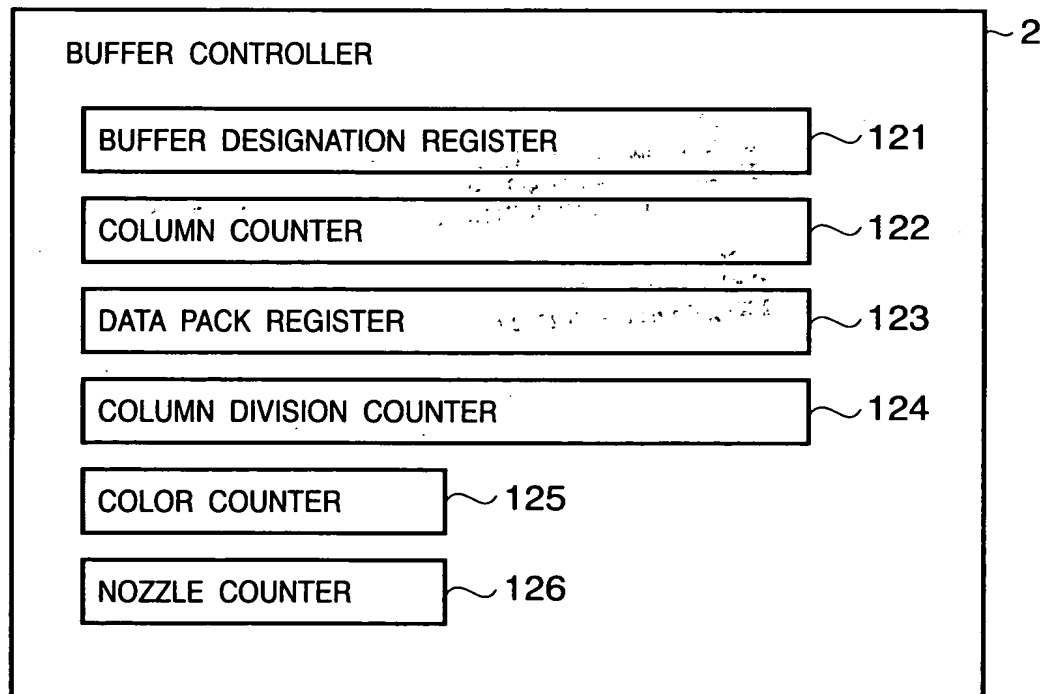
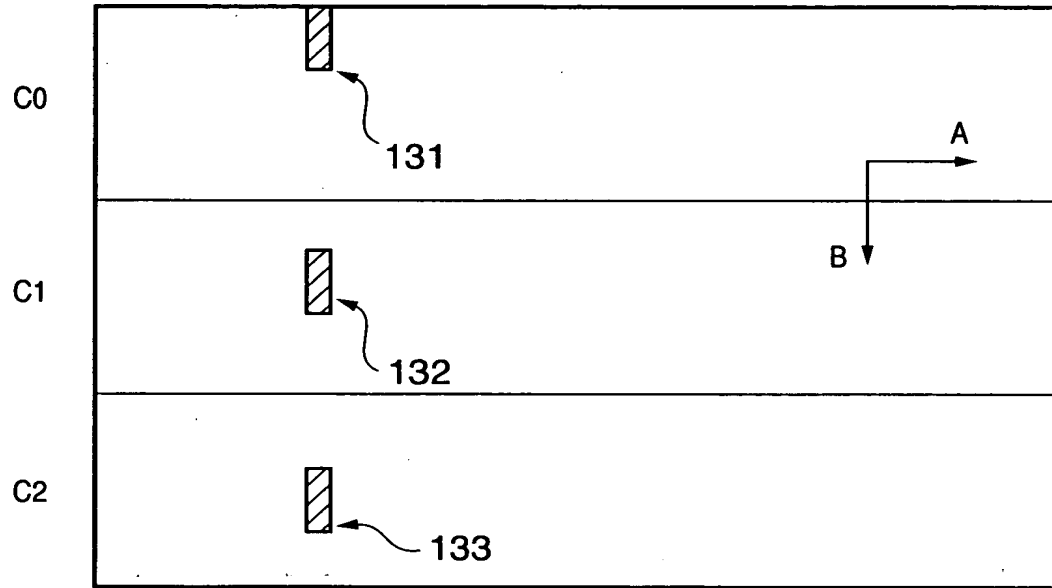


FIG. 13**FIG. 14**